Introduction to Digital Circuits and Computer Design

Notes #7

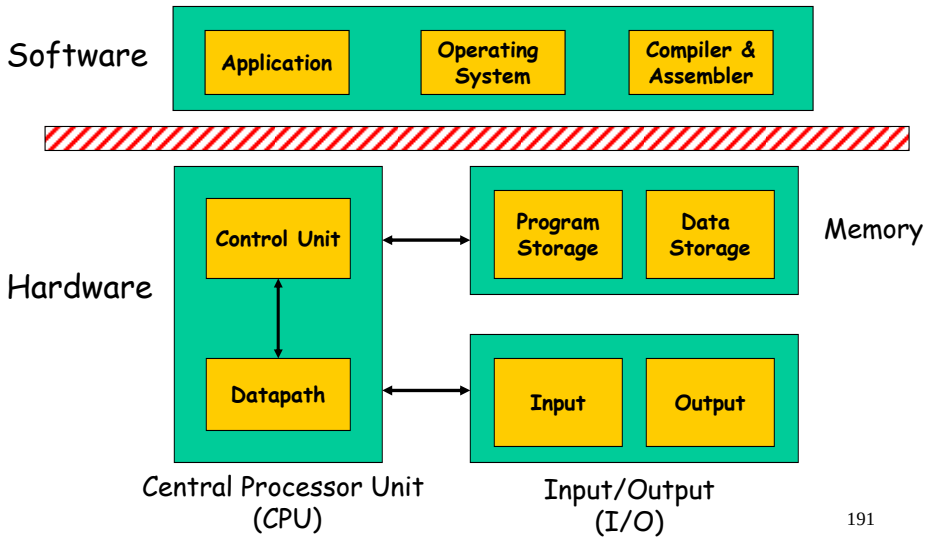
"The MicroComputer"

Spring Quarter 2010

Avinash Kodi

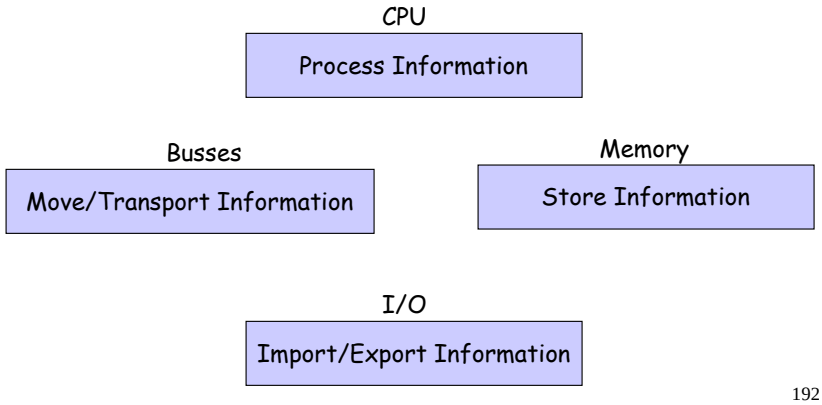
Acknowledgement: Dr. Maarten Uijt de Haag

Computer Architecture



Information!!!

Each of the Computer components has its own specific task with respect to information.



192

How

does the previous material (Notes #1,2,3,4,5) fit into computer design?

Datapath
Sequential logic: Registers
Combinational logic: MUXs, adders, multipliers, comparators.

Control Unit
Sequential logic: State machines, registers
Combinational logic: Control of datapath components

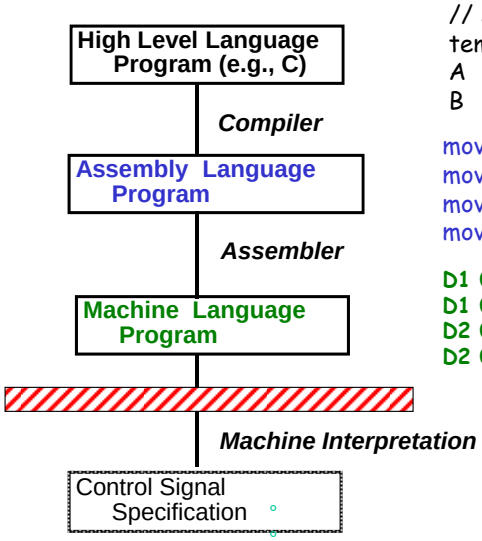
Memory
Sequential logic: Registers, RAM
Combinational logic: ROMs, decoders, multiplexers

I/O
Sequential logic: Latches (D-flip-flops)
Combinational logic: Decoders, multiplexers

193

Review (notes 1)

Information Representation Levels (top-down)




```
// Swap 'A' and 'B'
temp    = A;
A       = B;
B       = temp;
```

```
mov     AX,A
mov     BX,B
mov     A,BX
mov     B,AX
```

```
D1 00 00
D1 01 01   or
D2 00 01
D2 01 00
```

```
1101 0001 0000 0000 0000 0000
1101 0001 0000 0001 0000 0001
1101 0010 0000 0000 0000 0001
1101 0010 0000 0001 0000 0000
```

194

Review (notes 1)

Program: The Instruction Set

Instruction Set:
the language you use to talk to a specific computer
Or
a set of instructions used to tell a computer what functions/procedures to perform

Instruction Set Architecture:
what kind of instructions does the instruction set consist of.

Note: every computer has a different instruction set (Intel Pentium I, PowerPC G3, etc.)

If we talk about a computer in general and we want to describe the instructions any computer performs we can use **Register Transfer Language** (RTL);
A sort of universal language.

195

Review (notes 1)

Programming - RTL - Intro

- **RTL is machine independent !!!!**

(•) = "Contents of •"
← = "Move operand on right-hand-side to the left-hand-side destination"
🕑 = "AND "
🕓 = "OR"
🕗 = "Exclusive-OR"
'+' = "Addition"
'-' = "Subtraction"
'*' = "Multiplication"
'/' = "Division"

VHDL and Verilog are also machine independent description languages but they differ from RTL

196

A Simple Computer

Instruction Set Architecture Definition via RTL

Addition: (R1) ← (R1) + (R2)
(R1) ← (R1) + (M)

Load/Read: (R1) ← (M)
(R2) ← (M)

Write: (M) ← (R1)
(M) ← (R2)

Instructions

Q: suppose we want to identify the instructions by a unique number, how many bits do we need for that?

This computer has 2 general-purpose registers R1 and R2.

This computer also has a RAM memory, M.

197

Our Simple Computer

Op-code Selection - Example

Example:		Binary Code that identifies the instruction to the computer
Addition:	(R1) ← (R1) + (R2)	000
	(R1) ← (R1) + (M)	001
Load/Read:	(R1) ← (M)	010
	(R2) ← (M)	011
Write:	(M) ← (R1)	100
	(M) ← (R2)	101

This code is referred to as the op-code

198

Our Simple Computer

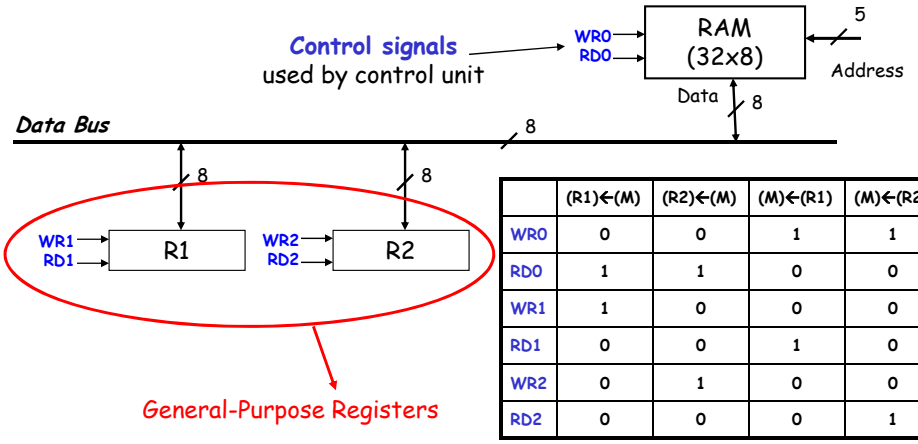
What do we need?

		Combinational	Sequential
Addition:	(R1) ← (R1) + (R2) (R1) ← (R1) + (M)	adder, multiplexer	registers, RAM
Load/Read:	(R1) ← (M) (R2) ← (M)		registers, RAM
Write:	(M) ← (R1) (M) ← (R2)		registers, RAM

199

Our Simple Computer

Step 1

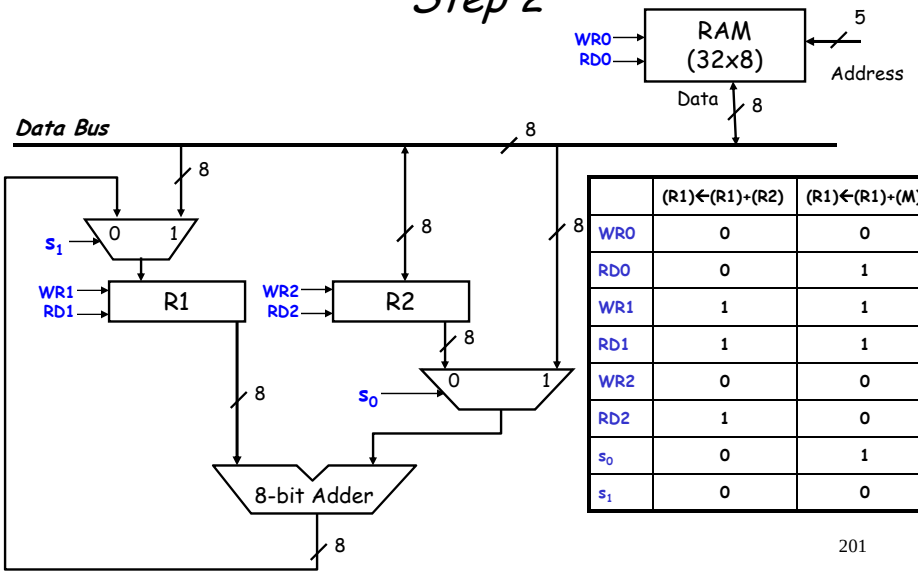


	(R1)←(M)	(R2)←(M)	(M)←(R1)	(M)←(R2)
WR0	0	0	1	1
RD0	1	1	0	0
WR1	1	0	0	0
RD1	0	0	1	0
WR2	0	1	0	0
RD2	0	0	0	1

200

Our Simple Computer

Step 2

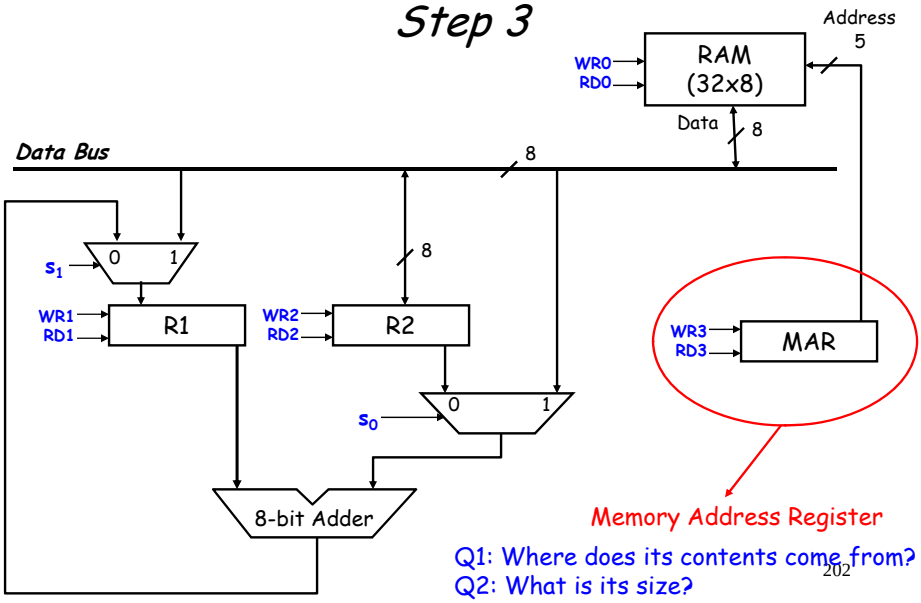


	(R1)←(R1)+(R2)	(R1)←(R1)+(M)
WR0	0	0
RD0	0	1
WR1	1	1
RD1	1	1
WR2	0	0
RD2	1	0
s_0	0	1
s_1	0	0

201

Our Simple Computer

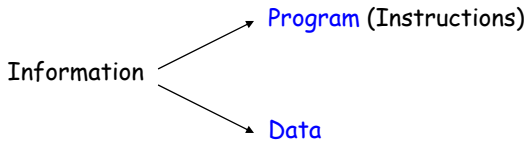
Step 3



Q1: Where does its contents come from?
Q2: What is its size?

Data & Programs

Storage

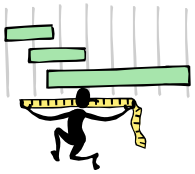


Q: Where is information stored?

Von Neumann architecture vs. Harvard architecture

Data Storage
Endians

- Big Endian:
  - Least Significant Byte @ higher address
  - Most Significant Byte @ lower address
- Little Endian:
  - Least Significant Byte @ lower address
  - Most Significant Byte @ higher address



- Little Endian processors: Intel 80x86, Pentium II, VAX, Alpha
- Big Endian processors: 680x0, Sun SPARC
- PowerPC: Alter the Endian-ness using instructions

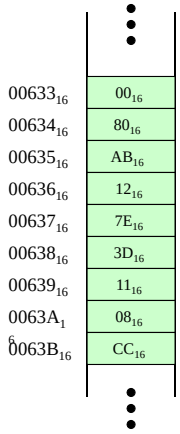
204

Data Storage
Example - Little Endian

Intel Pentium (little Endian)

Write $3D7E12AB_{16}$ to memory location 00635_{16} gives:

Read one 2-byte word from memory location 00639_{16} yields: 0811_{16}



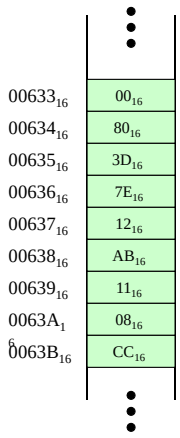
205

Data Storage
Example - Big Endian

SUN Sparc (Big Endian)

Write $3D7E12AB_{16}$ to memory location 00635_{16} gives:

Read one 2-byte word from memory location 00639_{16} yields: 1108_{16}



206

Data Storage
Bits & Bytes

Nibble	- 4 bits
Byte	- 8 bits
Word	- 16 bits (2 bytes)
Double word	- 32 bits (4 bytes)
Quad word	- 64 bits (8 bytes)

207

Program Storage

The Instruction

Op-code (operation code)

Determines what operation must be performed.
(what to do)

Operand

Determines on what variable (memory) the operation must be performed.
(what to do it with)

Either a register or a memory location.

208

Program Storage

The Instruction Format

0-address: <op-code>
1-address: <op-code> address1
2-address: <op-code> address1, address2
3-address: <op-code> address1, address2, address3

Either there is no operand at all or there is an implied operand within the op-code.

209

Program Storage
Instruction Examples

Processor	Assembly	RTL	Format
MIPS	add a, b, c	(a) ←(b) + (c)	3-address
Intel	mov ax,[1234h]	(ax) ←(1234h)	2-address
PIC	movlw 05	(w) ← 5	1-address
Motorola 68HC11	inca	(a) ←(a) + 1	0-address
Intel	hlt	halt computer	0-address

Unique for each computer

210

Instruction Set Architecture
Addressing Modes

An instruction consists of an op-code & an operand.
The operand represents the variable on which the instruction operates.

<op-code> address1, address2, …

This is referred as an address field.

The addressing modes describe how the operands are obtained from the instruction address fields.

211

Instruction Set Architecture

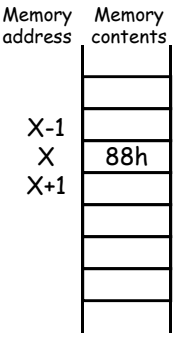
Immediate and Direct Addressing

Immediate Addressing:
The address field is the operand

RTL: **(R) ← 88**

Direct Addressing
The address field contains the address, X, of the memory location of the operand

RTL: **(R) ← (X)**



Example: (R1) ← (1234)
Move the contents of memory with address 'X' to register, **R1**

Note: 'X' can also be the name of another register: (R1) ← (R2)

212

Instruction Set Architecture

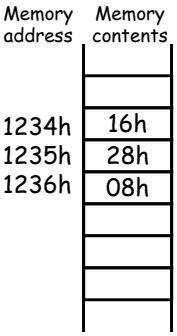
Example

Given the following RTL code segment:

Line 1: **(R1) ← 12h**
Line 2: **(R2) ← (1235h)**
Line 3: **(R3) ← (R1) + (R2)**
Line 4: **(R4) ← (R1) - (R1)**
Line 5: **(1236h) ← (R4)**

Questions:
What are the contents of R1, R2, R3, R4 and memory Location 1236h?

Given:



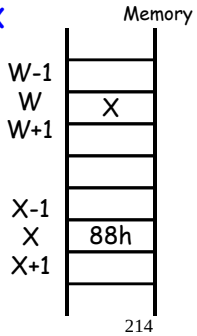
213

Instruction Set Architecture

Indirect Addressing

Indirect Addressing
The address field contains the address, W, of a memory location, which in turn contains the address, X, of the operand

RTL: **(R) ← ((W))** **with (W) = X**



Note: if W is a register, we refer to this addressing mode as **Register Indirect Addressing**

Example: if (R1) = 12h and (R2) = 45h then

(R1) ← ((R2))

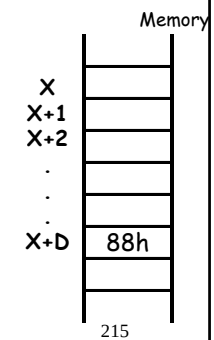
Moves the contents of memory location 45h to R1

Instruction Set Architecture

Relative Addressing

Relative Addressing
The address field contains a relative address, D (also called an offset or displacement). The instruction also implicitly or explicitly identifies another storage location L containing additional information.
The address is obtained by some function of L and D. (usually an addition)

RTL: **(R) ← ((L) + D)**



If L is a register, some computers use the nomenclature:
Register indirect with offset (if D = a number)

RTL: **(R) ← ((L) + (D))**

If L and D are registers, some computers use the nomenclature:
Register indirect with index (if D = a register)

Instruction Set Architecture
Example

Given the following RTL code segment:

Line 1: **(R1) ← ((R2) + 12h)**
Line 2: **(R3) ← ((R2) + (R1))**

And given that before the two lines of code:
(R2) = 1224h

Questions:
What are the contents of R1 and R3?

Given:

Memory address	Memory contents
1234_H	12h
1235_H	11h
1236_H	10h

Instruction Set Architecture
Effective Address

The effective address is the address of
the operand as calculated by the addressing mode.

RTL: (R) ← ((L) + D)

"(L) + D" is referred to as the effective address of this operand

RTL: (R) ← (X)

"X" is referred to as the effective address of this operand

Instruction Set Architecture

Relative Addressing - 2

Why is relative addressing useful?

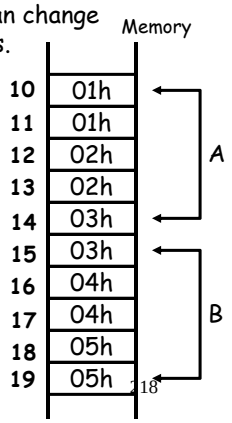
1). The instruction length may be reduced!

2). By changing L (often called the base register) one can change all the absolute addresses for a block of instructions.

Example:
a) (R1) ← ((L1) + D)
b) (R2) ← ((L1) - D)
c) (R1) ← (R1) + (R2)

A: Suppose (L1) = 12, D = 2, then
after a: (R1) = 3, after b: (R2) = 1
after c: (R1) = 4

B: Suppose (L1) = 17, D = 2, then
after a: (R1) = 5, after b: (R2) = 3
after c: (R1) = 8



Instruction Set Architecture

Relative Addressing - 3

Why is relative addressing useful (continued)?

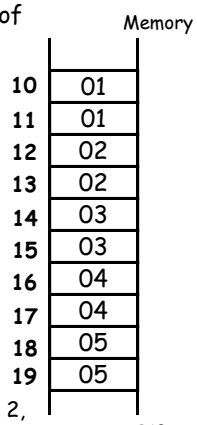
3). L can store indexes to facilitate processing of indexed data such as tables and lists

Example:
(L) ← 0
(R1) ← 0
while (L) is not equal to 5,
    (R1) ← (R1) + ((L) + 10)
    (L) ← (L) + 1
end

Condition

Each time you execute these lines of RTL is referred to as a pass

While loop



Q: What is (R1) after execution of these RTL instructions?

Answer: after pass 1: (R1) = 01, (L) = 1, after pass 2: (R1) = 02, (L) = 2, after pass 3: (R1) = 04, (L) = 3, after pass 4: (R1) = 06, (L) = 4, after pass 5: (R1) = 09, (L) = 5

Instruction Set Architecture

Relative Addressing - 4

Drawback?

Relative addressing requires more combinational and sequential logic to implement!

RTL: **(R) ← ((L) + D)**

220

Addressing Modes

Processors Examples

- PowerPC:
  - Register indirect with offset,
  - Register indirect with index.
- MIPS RX000:
  - Register indirect with offset.
- PICMicro 16F877:
  - Direct,
  - Indirect,
  - Relative.
- Intel:
  - Register indirect and direct
  - Immediate
  - Based addressing, based-indexed addressing
  - Direct, etc.

221